Programación avanzada en Python

Lección 3: Programación y manejo de funciones

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Programación y manejo de funciones

PRESENTACIÓN Y OBJETIVOS

En este tercer capítulo vamos a ver cómo trabajar con funciones, para ello empezaremos viendo cómo declararlas y seguidamente, veremos cómo enviar valores a dichas funciones mediante argumentos y parámetros. Estudiaremos cómo retornar valores desde una función y, por último, veremos cómo controlar errores mediante excepciones.

Objetivos

- En esta lección aprenderás a:
- Crear funciones en Python
- Trabajar con argumentos y parámetros
- Controlar errores con excepciones

1. FUNCIONES

La función es un conjunto de códigos que se ejecutan cuando se le llama y realiza una determina acción. Las funciones se utilizan principalmente para hacer el programa más compacto y cuando un conjunto particular de códigos quiere ser reutilizado. Dando diferentes valores de entrada una función puede reutilizarse muchas veces y tomar la salida basada en los valores dados. Si se definen correctamente, el programa puede ser más flexible.

1.1 Definición de funciones

En Python una función se define con la palabra clave '*def*'.

Sintaxis:

def nombre_función () :

declaración1

declaración2

.....

El nombre de la función se escribe con la palabra clave *def,* seguido del nombre de la función, el paréntesis y los dos puntos. La sangría debe ser adecuada para ejecutar las declaraciones dentro de la función.

Así es como se define un conjunto de códigos bajo un nombre. En el programa se definen primero las funciones y se las llama por el nombre cuando queramos hacer uso de ella.

La llamada a la función con su nombre sería:

Función_nombre()

Así que cuando se llama a la función predefinida, el control se mueve a la definición de la función y luego entra y ejecuta las declaraciones dentro de la función.

Veamos un ejemplo:

```
def new_function():
  print("Hai")
```

Esta función declara en el ejemplo anterior se llamaría de la siguiente manera:

```
new_function()

Hai
```

Cuando se llama a la función, el controlador se mueve a la definición de la función, entra en ella y se imprime Hai.

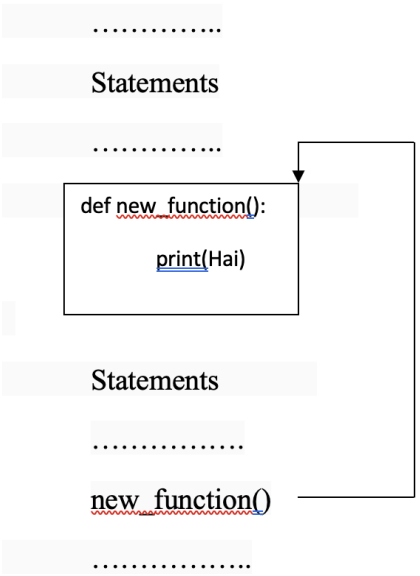

Figura 1.1 Funcionamiento función

Una función puede ser llamada tantas veces como sea necesario a lo largo del programa.

1.2 Envío de valores

Una función puede tomar los datos como entrada a través de sus paréntesis, estos valores se llaman parámetros. Los parámetros se declaran en el momento de la definición de la función.

En el ejemplo anterior, hay una función simple para imprimir un texto. Podemos cambiar la función para imprimir algo dado por el usuario a través de parámetros.

```
def example_fun(name):
  print("hello",name)
```

Aquí name es el parámetro y cuando se llama a la función, la entrada se da a través del paréntesis.

```
example_fun('xyz')
```

```
hello xyz
```

Por lo que xyz se pasa como nombre, entra en la función e imprime la entrada del parámetro.

Una función puede ser llamada tantas veces como se necesite a lo largo del programa, con diferentes entradas por parámetros.

En el ejemplo anterior, la función 'example_fun' puede ser llamada muchas veces con los nombres deseados como parámetros de entrada.

```
example_fun('abc')
example_fun('qaz')
example_fun('pqr')
```

```
hello abc
hello qaz
hello pqr
```

No hay límite en el número de parámetros. Puede ser más de uno y deben estar definidos de una manera ordenada.

```
def fun_parameters(name, age):
  print('I am', name,"& my age is",age,)
```

```
fun_parameters('xyz',55)
```

```
I am xyz & my age is 55
```

Los valores del parámetro siempre se envían de forma ordenada. En el ejemplo anterior el primer parámetro xyz pasa como primer parámetro 'name' y así sucesivamente.

Es posible dar los parámetros en diferente orden especificando el valor de cada parámetro. Se puede especificar de la siguiente manera:

nombre_parámetro = valor

```
def my_function(name3, name2, name1):
  print("The third one is " + name3)

my_function(name1 = "xyz", name2 = "abc", name3 = "def")
```

```
The third one is def
```

1.3 Retornando valores

Las funciones pueden devolver o retornar una salida. La palabra clave 'return' se utiliza en la función para devolver dichos valores

Veamos la sintaxis:

def fun_return ():

    Statements

    return value/expression/variables/strings etc.

value = fun_return()

Consideremos la siguiente función:

```
def funct_return(name, age):
  print('hello I am', name,",",age,"years old")
  result = age + 10
  return result
```

En el anterior ejemplo, la función funct_return toma 2 entradas y da 1 salida. El 'nombre' y la 'edad' son dos parámetros que se pueden dar a la función en el momento de la llamada y después de realizar algunos cálculos, devulve la variable 'result' al lugar donde se ha llamado.

En este caso, el valor de retorno puede ser almacenado o directamente impreso cuando la función es llamada:

```
print(funct_return('xyz',55))
```

```
hello I am xyz , 55 years old
65
```

Aquí se imprime directamente el valor de la edad que da 55.

O podemos asignar el valor devuelto a la variable y luego imprimir o hacer otros cálculos.

```
age = funct_return('xyz',55)
print("after 10 years, my age become", age)
```

```
hello I am xyz , 55 years old
after 10 years, my age become 65
```

Funcionamiento:

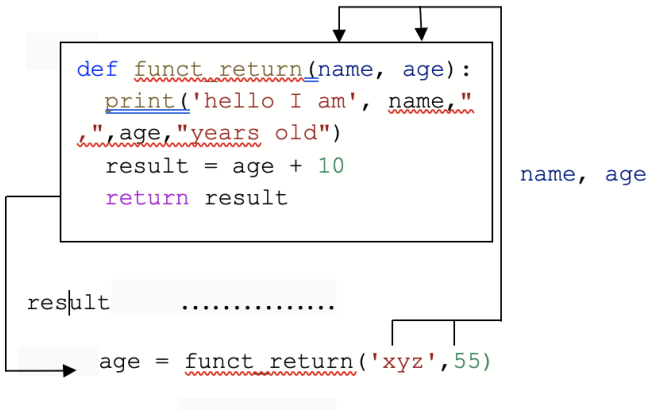

Figura 1.2 Retornando valores

Una función puede devolver datos en cualquier tipo de estructura de datos, como una lista, una tupla, etc.

```
def funct_list(input_list):
  name = 'xyz'
  age = 55
  height = 156.5
  return [name,age,height]
```

Aquí toma una lista y devuelve los valores como una lista. La lista puede ser inicializada y pasada a la función, finalmente la lista será devuelta.

```
new_list = []
new_list = funct_list(new_list)
print(new_list,type(new_list))
```

```
['xyz', 55, 156.5] <class 'list'>
```

Una función no debería estar vacía. Pero habrá algunas situaciones en las que tenga sentido crearla como vacía. Una función se puede hacer vacía con la palabra clave 'pass'.

```
def function():
  pass
```

Cuando se ejecuta la función, no devolverá ningún error.

Retornando valores con condiciones

Una función puede devolver el valor basándose en algunas condiciones:

```
def conditions(x,y):
  if x>y:
    return x-y
  else:
    return y-x
```

Aquí hay dos *return* en la función, pero sólo uno se utilizará basado en la condición.

```
print(conditions(10,5))
```

```
5
```

Si la condición es verdadera, el primer *return* funcionará.

Por otro lado, no es obligatorio devolver un valor :

```
print(no_return(5,4))
```

```
None
```

La suma de x e y se calcula pero no devuelve nada

1.4 Argumentos y parámetros

Los términos argumentos y parámetros se utilizan casi con el mismo propósito. La principal diferencia es:

Parámetros - son los valores definidos en la función en el momento de la definición

Argumentos - son los valores dados por el usuario y pasados a las funciones

Veamos un ejemplo:

```
def fun_parameters(name, age):
  print('I am', name,"& my age is",age,)
```

La función anterior se llamaría de la siguiente manera:

```
fun_parameters('xyz',55)
```

```
I am xyz & my age is 55
```

De este ejemplo los argumentos y parámetros se pueden especificar como:

Parámetros - nombre y edad se definen en la definición de la función

Argumentos - xyz & 55 son valores que se pasan como parámetro a la función.

Las funciones retornando funciones

Una función puede devolver otra función ya definida dentro de la función.

def función_externa(param1):

    declaraciones

    def función_interna(param2):

        declaraciones

        return algo

    .....

    return función_interna

se puede llamar como

una_variable = función_externa(param1)

una_variable(param2)

Veamos un ejemplo:

```
def outer_funct(x):
    def inner_funct(y):
        return x + y

    return inner_funct
```

Este es un ejemplo con función externa, *outer_funct* con parámetro x y dentro de la función hay una *inner_funct* con parámetro y

La función interna devuelve la suma de x e y, la función externa devuelve la función interna. Así que cuando llamamos a la función externa devuelve el objeto de la función interna y vuelve a llamar a la función interna y calcula el resultado

```
add = outer_funct(15)

print("The result is", add(10))
```

```
The result is 25
```

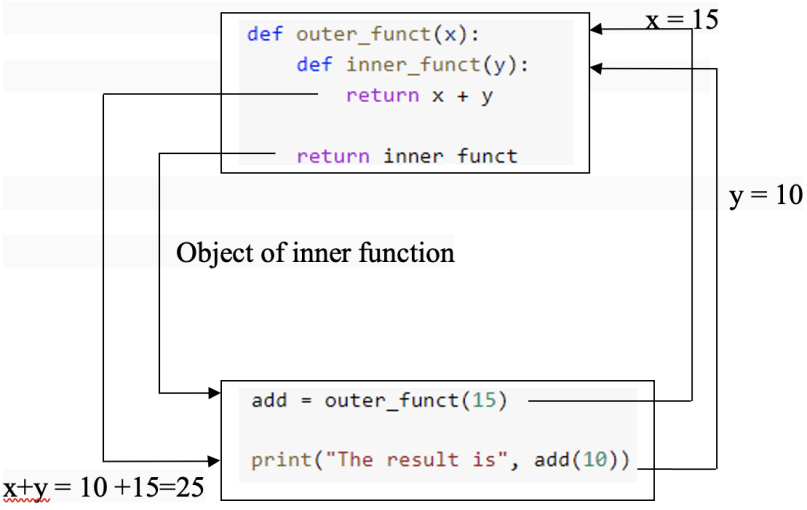

Figura 1.3 Funciones retornando funciones

Funciones recursivas

Si una función devuelve la misma función entonces se llama recursión. El control se mantiene como una rotación hasta satisfacer una condición o con una declaración de ruptura.

```
def addition(x):
  if x !=0:
    print(x)
    return addition(x - 1)
```

La función de *addition* devuelve la misma función con x-1 y luego entra de nuevo en la función y continúa así con cada iteración, hasta que la condición de que x no sea igual a 1 se cumpla.

```
addition(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

Finalmente imprime los valores cuando x se convierte en 0

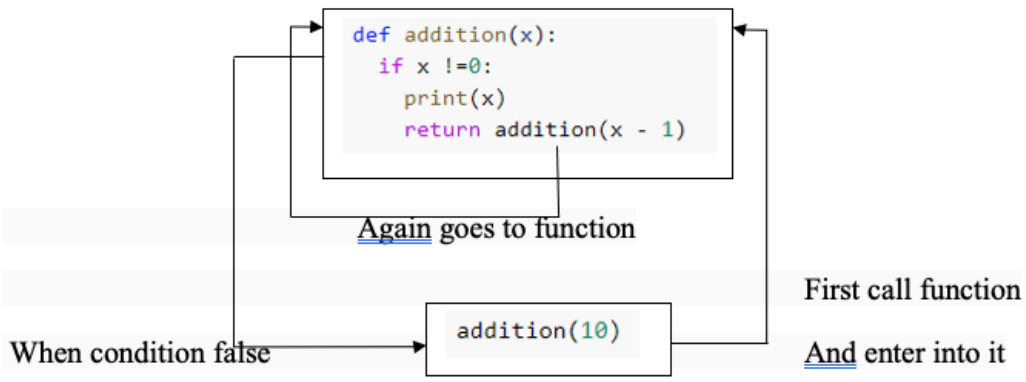

Figura 1.4 Funciones recursivas

1.5 Excepciones

Si los programas Python terminan con un error, finalizarán y mostrarán el error en la terminal. En los programas Python, los posibles errores pueden ser capturados usando algunos objetos que se llaman excepciones.

Las excepciones pueden ser manejadas por un bloque de excepción llamado '*try - except*'

Sintaxis:

```
try:
        expresiones/declaraciones
except:
        print error
```

Las sentencias que se encuentre dentro del *try*, serán comprobadas para detectar si hay un error o no en las sentencias. Si hay un error entonces la parte except se activará e imprimirá el posible error que se menciona en el bloque.

Veamos un ejemplo:

```
x = ‘5’
y = 5
x+y
```

En este código se muestra un error al sumar la cadena con el entero por lo que debemos convertir el tipo de dato antes de sumarlo.

Normalmente, el programa terminará con este error. Este final forzado, se puede evitar y eliminar el error usando el try- except, para que el programa se ejecute sin problemas e imprima el error simplemente.

```
x = '5'
y = 5
try:
  x+y
except:
  print("Error")
```

```
Error
```

Si no hay error no se imprimirá la declaración en *except*. Hay otro bloque que se utiliza como una parte *else* para *except* que es '*finally*'.

Si no hay ningún error después de probar las sentencias en el *try* entonces pasará a la parte del *finally*.

En el siguiente ejemplo, ambas variables son cadenas y la concatenación se producirá con el símbolo +. Así que no hay error y la parte final se activará.

```
x = '5'
y = '5'
try:
  x+y
except:
  print("Error")
finally:
  print('No error')
```

```
No error
```

En *try- except*, se pueden capturar cualquier tipo de error como: error de tipo, error de nombre, error de valor, etc.

Aserciones

En Python, los errores en la lógica del programa pueden ser señalados con declaraciones de aserción. El error de aserción puede ser capturado con la palabra clave assert:

assert expression, arguments

Veamos un ejemplo:

```
x = -2
if (x>0):
  print("positive")
```

Aquí la condición if es falsa porque x es negativa. Así que cuando la x no sea positiva, podemos establecer un error de aserción para hacer que sea positiva:

```
x = -2
assert (x > 0), "x is negative number"
```

Cuando ejecutamos esto se comprueba la condición dentro de la aserción y el argumento se muestra con el error de aserción

```
x = -2
assert (x > 0), "x is negative number"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-18-fffe381299fa> in <module>()
      1 x = -2
----> 2 assert (x > 0), "x is negative number"

AssertionError: x is negative number
```

2. PUNTOS CLAVE

- Las **funciones** se utilizan principalmente para hacer el programa más compacto.
- Una función puede tomar los datos como entrada a través de sus paréntesis, estos valores se llaman **parámetros**.
- Las funciones pueden devolver o retornar una salida, mediante la palabra reserva **return.**
- Los **parámetros** son los valores definidos en la función en el momento de la definición.
- Los **argumentos** son los valores dados por el usuario y pasados a las funciones.
- Los posibles errores pueden ser capturados usando algunos objetos que se llaman **excepciones**.